(72) Inventors:
• Chaddha, Navin
Stanford, California 94305 (US)
• Northcutt, J. Duane
Sunnyvale, California 94087 (US)
• Wall, Gerald A.
San Jose, California 95136 (US)
• Hanko, James G.
Redwood City, California 94061 (US)

(74) Representative: Cross, Rupert Edward Blount
BOULT, WADE & TENNANT
27 Furnival Street
London EC4A 1PQ (GB)

(54) Encoder for an end-to-end scalable video delivery system

(57) A software-based encoder is provided for an end-to-end scalable video delivery system that operates over heterogeneous networks. The encoder utilizes a scalable video compression algorithm based on a Laplacian pyramid decomposition to generate an embedded information stream. The decoder decimates a highest resolution original image, e.g., 640x480 pixels, to produce an intermediate 320x240 pixel image that is decimated to produce an intermediate 160x120 pixel image that is compressed to form an encodable base layer 160x120 pixel image. This base layer image is decompressed to form an image that is up-sampled by interpolation to produce an up-sampled 320x240 pixel image. This up-sampled image is subtracted from the intermediate 320x240 pixel image to form an error image that is compressed and encoded as a first enhancement 640x480 pixel layer. The decompressed base layer image is also up-sampled at step to produce an up-sampled 640x480 pixel image that is subtracted from the original 640x480 pixel image 200 to yield an error image that is compressed to yield a second enhancement 320x240 pixel layer. Collectively, the base and enhancement layers comprise the transmitted embedded bit stream. At the receiving end, the decoder extracts from the embedded stream different streams at different spatial and temporal resolutions. Because decoding requires only additions and look-ups from a small stored table, decoding occurs in real-time.
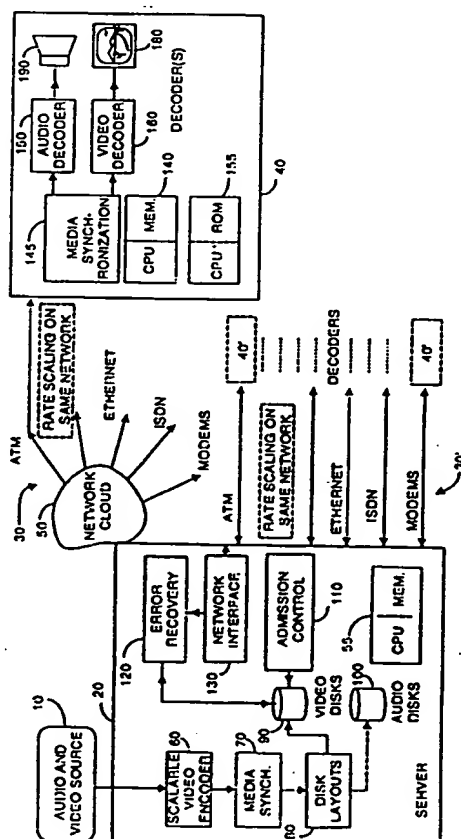
FIGURE 1

EP 0 739 140 A2

## FIELD OF THE INVENTION

5      The present invention relates generally to video delivery systems, and more specifically to encoders for such systems that will permit video to be delivered scalably, so as to maximize use of network resources and to minimize user-contention conflicts.

## BACKGROUND OF THE INVENTION

10

It is known in the art to use server-client networks to provide video to end users, wherein the server issues a separate video stream for each individual client.

A library of video sources is maintained at the server end. Chosen video selections are signal processed by a server encoder stored on digital media, and are then transmitted over a variety of networks, perhaps on an basis that allows a remote viewer to interact with the video. The video may be stored on media that includes magnetic disk, CD-
15     ROM, and the stored information can include video, speech, and images. As such, the source video information may have been stored in one of several spatial resolutions (e.g., 160x120, 320x240, 640x480 pixels), and temporal resolutions (e.g., 1 to 30 frames per second). The source video may present bandwidths whose dynamic range can vary from 10 Kbps to 10 Mbps.

20     The signal processed video is transmitted to the clients (or decoders) over one or more delivery networks that may be heterogeneous, e.g., have widely differing bandwidths. For example, telephone delivery lines can transmit at only a few tens of Kbps, an ISDN network can handle 128 Kbps, ethernet at 10 Mbps, whereas ATM networks handle even higher transmission rates.

Although the source video has varying characteristics, prior art video delivery systems operate with a system
25     bandwidth that is static or fixed. Although such system bandwidths are fixed, in practice, the general purpose computing environment associated with the systems are dynamic, and variations in the networks can also exist. These variations can arise from the outright lack of resources (e.g., limited network bandwidth and processor cycles), contention for available resources due to congestion, or a user's unwillingness to allocate needed resources to the task.

Prior art systems tend to be very computationally intensive, especially with respect to decoding images of differing
30     resolutions. For example, where a prior art encoder transmits a bit stream of, say, 320x240 pixel resolution, but the decoder requires 160x120 pixel resolution, several processes must be invoked, involving decompression, entropy coding, quantization, discrete cosine transformation and down-sampling. Collectively, these steps require too long to be accomplished in real-time.

Color conversions, e.g., YUV-to-RGB are especially computationally intensive, in the prior. In another situation,
35     an encoder may transmit 24 bits, representing 16 million colors, but a recipient decoder may be coupled to a PC having an 8 bit display, capable of only 256 colors. The decoder must then dither the incoming data, which is a computationally intensive task.

Unfortunately, fixed bandwidth prior art systems cannot make full use of such dynamic environments and system variations. The result is slower throughput and more severe contention for a given level of expenditure for system
40     hardware and software. When congestion (e.g., a region of constrained bandwidth) is present on the network, packets of transmitted information will be randomly dropped, with the result that no useful information may be received by the client.

Video information is extremely storage intensive, and compression is necessary during storage and transmission. Although scalable compression would be beneficial, especially for browsing in multimedia video sources, existing com-
45     pression systems do not provide desired properties for scalable compression. By scalable compression it is meant that a full dynamic range of spatial and temporal resolutions should be provided on a single embedded video stream that is output by the server over the network(s). Acceptable software-based scalable techniques are not found in the prior art. For example, the MPEG-2 compression standard offers limited extent scalability, but lacks sufficient dynamic range of bandwidth, is costly to implement in software, and uses variable length codes that require additional error correction
50     support.

Further, prior art compression standards typically require dedicated hardware at the encoding end, e.g., an MPEG board for the MPEG compression standard. While some prior art encoding techniques are software-based and operate without dedicated hardware (other than a fast central processing unit), known software-based approaches are too computational intensive to operate in real-time. For example, JPEG software running on a SparcStation 10 workstation
55     can handle only 2-3 frames/second, e.g., about 1% of the frame/second capability of the present invention.

Considerable video server research in the prior art has focussed on scheduling policies for on-demand situations, admission control, and RAID issues. Prior art encoder operation typically is dependent upon the characteristics of the client decoders. Simply stated, relatively little work has been directed to video server systems operable over hetero-

geneous networks having differing bandwidth capabilities, where host decoders have various spatial and temporal resolutions.

In summary, there is a need for a video delivery system that provides end-to-end video encoding such that the server outputs a single embedded data stream from which decoders may extract video having different spatial resolutions, temporal resolutions and data rates. The encoder should be software-based and provide video compression that is bandwidth scalable, and thus deliverable over heterogeneous networks whose transmission rates vary from perhaps 10 Kbps to 10 Mbps. Such a system should accommodate lower bandwidth links or congestion, and should permit the encoder to operate independently of decoder capability or requirements.

The decoder for such system should be software-based (e.g., not require specialized dedicated hardware beyond a computing system) or should be implemented using inexpensive read-only memory type hardware, and should permit real-time decompression. The system should permit user selection of a delivery bandwidth to choose the most appropriate point in spatial resolution, temporal resolution, data-rate and in quality space. The system should also provide subjective video quality enhancement, and should include error resilience to allow for communication errors.

The present invention provides a software-based encoder for such a system.

## SUMMARY OF THE INVENTION

The present invention provides a software-based server-encoder for an end-to-end scalable video delivery system, wherein the server-encoder operates independently of the capabilities and requirements of the software-based decoder(s). The encoder uses a scalable compression algorithm based upon Laplacian pyramid decomposition. An original 640x480 pixel image is decimated to produce a 320x240 pixel image that is itself decimated to yield a 160x120 pixel base image that is encoder-transmitted.

This base image is then compressed to form a 160x120 pixel base layer, that is decompressed and up-sampled to produce an up-sampled 320x240 pixel image. The up-sampled 320x240 pixel image is then subtracted from the 320x240 pixel image to provide an error image that is compressed as transmitted as a first enhancement layer. The 160x120 pixel decompressed image is also up-sampled to produce an up-sampled 640x480 pixel image that is subtracted from the original 640x480 pixel image to yield an error image that is compressed and transmitted as a second enhancement layer.

Collectively the base layer, and first and second enhancement layers comprise the single embedded bitstream that may be multicast over heterogeneous networks that can range from telephone lines to wireless transmission. Packets within the embedded bit-stream preferably are prioritized with bits arranged in order of visual importance. The resultant bit stream is easily rescaled by dropping less important bits, thus providing bandwidth scalability dynamic range from a few Kbps to many Mbps. Further, such embedded bit stream permits the server system to accommodate a plurality of users whose decoder systems have differing characteristics. The transmitting end also includes a market-based mechanism for resolving conflicts in providing an end-to-end scalable video delivery service to the user.

At the receiving end, decoders of varying characteristics can extract different streams at different spatial and temporal resolutions from the single embedded bit stream. Decoding a 160x120 pixel image involves only decompressing the base layer 160x120 pixel image. Decoding a 320x240 pixel image involves decompressing and up-sampling the base layer to yield a 320x240 pixel image to which is added error data in the first enhancement layer following its decompression. To obtain a 640x480 pixel image, the decoder up-samples the up-sampled 320x240 pixel image, to which is added error data in the second enhancement layer, following its decompression. Thus, decoding is fast and requires only table look-ups and additions. Subjective quality of the compressed images is enhanced using perceptual distortion measures. The system also provides joint-source channel coding capability on heterogenous networks.

Other features and advantages of embodiments of the invention will appear from the following description in which the preferred embodiments have been set forth in detail, in conjunction with the accompanying drawings.

## BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a block diagram of an end-to-end scalable video system, in which the present invention may be embodied;

FIGURE 2 is a block/flow diagram depicting a software-based encoder that generates a scalable embedded video stream, embodying the present invention;

FIGURE 3 is a block/flow diagram depicting a decoder recovery of scalable video from a single embedded video stream.

## DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Figure 1 depicts an end-to-end scalable video delivery system, including a software-based encoder embodying the present invention. A source of audio and video information 10 is coupled to a server or encoder 20. The encoder signal processes the information to produce a single embedded information stream that is transmitted via homogeneous networks 30, 30' to one or more target clients or software-based decoder systems 40, 40', which decoder uses minimal central processor unit resources. Network transmission may be through a so-called network cloud 50, from which the single embedded information stream is multicast to the decoders, or transmission to the decoders 40' may be point-to-point.

The networks are heterogeneous in that they have widely varying bandwidth characteristics, ranging from as low as perhaps 10 Kbps for telephones, to 100 Mbps or more for ATM networks. As will be described, the single embedded information stream is readily scaled, as needed, to accommodate a lower bandwidth network link or to adapt to network congestion.

Server 20 includes a central processor unit ("CPU") with associated memory, collectively 55, a scalable video encoder 60, according to the present invention, a mechanism 70 for synchronizing audio, video and textual information, a mechanism 80 for arranging the information processed by the scalable video encoder onto video disks 90 (or other storage media). Storage 100 is also provided for signal processed audio information. Software comprising the scalable video encoder 60 preferably is digitally stored within server 20, for example, within the memory associated with CPU unit 55.

An admission control mechanism 110 is coupled to the processed video storage 90, as is a communication error recovery mechanism 120 for handling bit errors or packet cell loss. The decoder algorithm provides error resilience allowing for such communication errors. The server communicates with the heterogeneous network(s) through a network interface 130.

Scalable video encoder 60 differs from the prior art in that it preferably is implemented in software only (e.g., no dedicated hardware), and generates a single embedded information stream. Encoder 60 employs a new video coding algorithm based on a Laplacian pyramidal decomposition to generate the embedded information stream. (Laplacian pyramids are a form of compression known to those skilled in the art, and for that reason further details are not presented here.) The generated embedded stream allows server 20 to host decoders 40, 40' having various spatial and temporal resolutions, without the server having to know the characteristics of the recipient decoder(s).

With reference to Figure 2, an original 640x480 pixel image 200 from source 10 is coupled to the scalable video encoder 60. At process step 210, this image is decimated (e.g., filtered and sub-sampled) to 320x240 pixels (image 220), step 210, and at process step 230 image 220 is decimated to produce a base layer 160x120 pixel image 240 for encoding by encoder 60.

For the 160x120 pixel base layer, encoding preferably is done on a 2x2 blocks (e.g., two adjacent pixels on one line, and two adjacent pixels on a next line defining the block) with DCT followed by tree-structured vector quantization ("TSVQ") of the results of that transform. For the 320x240 first enhancement layer, encoding is done on 4x4 blocks, with DCT followed by TSVQ, and for the 640x480 pixel enhancement layer, encoding is done on 8x8 blocks, again with DCT followed by TSVQ.

At step 250, the 160x120 pixel base image 240 is compressed to form a 160x120 pixel base layer 260 and then at step 270 is decompressed. The resulting decompressed image 280 is up-sampled by interpolation step 290 to produce an up-sampled 320x240 pixel image 300.

At summation step 310, the up-sampled 320x240 pixel image 300 is subtracted from the 320x240 pixel image 220 to give an error image 320. At step 330 the error image 320 is compressed and then transmitted as a first enhancement 640x480 pixel layer 340.

The 160x120 pixel decompressed image 280 is also up-sampled at step 350 to produce an up-sampled 640x480 pixel image 360. At summation step 370, the up-sampled 640x480 pixel image 360 is subtracted from the original 640x480 pixel image 200 to yield an error image 380. At step 390, the error image 380 is compressed to yield a second enhancement 320x240 pixel layer 400 that is transmitted. Collectively, layers 260, 340 and 400 comprise the embedded bit-stream generated by the scalable video encoder 60.

Thus, it is appreciated from Figure 2 that a scalable video encoder 60 according to the present invention encodes three image resolutions. The transmitted base layer 260 has compressed data for the compressed 160x120 pixel image 240. The first enhancement layer 340 has error data for the compressed 320x240 pixel image 220, and the second enhancement layer 400 has error data for the compressed 640x480 pixel image 200.

The present embodiment uses vector quantization across transform bands to embed coding to provide bandwidth scalability with an embedded bit stream. Vector quantization techniques are known in the art. See, for example, A. Gerso and R. M. Gray, "Vector Quantization and Signal Compression", Kluwer Academic Press, 1992.

Embedded coding and vector quantization may each be performed by tree-structured vector quantization methods ("TSVQ"), e.g., by a successive approximation version of vector quantization ("VQ"). In ordinary VQ, the codewords

lie in an unstructured codebook, and each input vector is mapped to the minimum distortion codeword.

Thus, VQ induces a partition of a input space into Voronoi encoding regions.

By contrast, when using TSVQ, the codewords are arranged in a tree structure, and each input vector is successively mapped (from the root node) to the minimum distortion child node. As such, TSVQ induces a hierarchical partition, or refinement of the input space as three depth of the tree increases. Because of this successive refinement, an input vector mapping to a leaf node can be represented with high precision by the path map from the root to the leaf, or with lower precision by any prefix of the path.

Thus, TSVQ produces an embedded encoding of the data. If the depth of the tree is $R$ and the vector dimension is $k$, then bit rates $O/k$,......., $R/k$ can all be achieved. To achieve further compression, the index-planes can be run-length coded followed by entropy coding. Algorithms for designing TSVQs and its variants have been studied extensively. The Gerso and Grey treatise cited above provides a background survey of such algorithms.

In the prior art, mean squared error typically is used as distortion measure, with discrete cosine transforms ("DCT") being followed by scalar quantization. By contrast, the present embodiment performs DCT after which whole blocks of data are subjected to vector quantization, preferably with a perception model.

Subjectively meaningful distortion measures are used in the design and operation of the TSVQ. For this purpose, vector transformation is made using the DCT. Next, the following input-weighted squared error is applied to the transform coefficients:

$$d_{\underline{\imath}}(Y,\hat{y}) = \sum_{j=1}^{K} w_{\underline{\imath}}(Y_{\underline{\imath}} - \hat{Y}_{\underline{\imath}})^2$$

In the above equation, $y_j$ and $\hat{y}_j$ are the components of the transformed vector $y$ and of the corresponding reproduction vector $\hat{y}$, whereas $w_j$ is a component of the weight vector depending in general on $y$. Stated differently, distortion is the weighted sum of squared differences between the coefficients of the original transformed vector and the corresponding reproduced vector.

In the described arrangement, the weights reflect human visual sensitivity to quantization errors in different transform coefficients, or bands. The weights are input-dependent to model masking effects. When used in the perceptual distortion measure for vector quantization, the weights control an effective stepsize, or bit allocation, for each band. When the transform coefficients are vector quantized with respect to a weighted squared error distortion measure, the role played by weights $w_1,...,$ $w_K$ corresponds to stepsizes in the scalar quantization case. Thus, the perceptual model is incorporated into the VQ distortion measure, rather than into a stepsize or bit allocation algorithm, This permits the weights to vary with the input vector, while permitting the decoder to operate without requiring the encoder to transmit any side information about the weights.

In the first stage of the compression encoder shown in Figure 2, an image is transformed using DCT. The second stage of the encoder forms a vector of the transformed block. Next, the DCT coefficients are vector quantized using a TSVQ designed with a perceptually meaningful distortion measure. The encoder sends the indices as an embedded stream with different index planes. The first index plane contains the index for the rate 1/k TSVQ codebook. The second index plane contains the additional index which along with the first index plane gives the index for the rate 2/k TSVQ codebook. The remaining index planes similarly have part of the indices for 3/k, 4/k, ......,R/k TSVQ codebooks, respectively.

Such encoding of the indices advantageously produces an embedded prioritized bitstream. Thus, rate or bandwidth scalability is easily achieved by dropping index planes from the embedded bit-stream. At the receiving end, the decoder can use the remaining embedded stream to index a TSVQ codebook of the corresponding rate.

Frame-rate scalability can be easily achieved by dropping frames, as at present no interframe compression is implemented in the preferred embodiment of the encoder algorithm. The algorithm further provides a perceptually prioritized bit-stream because of the embedding property of TSVQ. If desired, motion estimation and/or conditional replenishment may also be incorporated into the system.

Scalable compression is also important for image browsing, multimedia applications, transcoding to different formats, and embedded television standards. By prioritizing packets comprising the embedded stream, congestion due to contention for network bandwidth, central processor unit ("CPU") cycles, etc., in the dynamic environment of general purpose computing systems can be overcome by intelligently dropping less important packets from the transmitted embedded stream.

Information layout on the video disk storage system 90 (see Figure 1) preferably involves laying the video as two streams, e.g., the base layer and the first and second enhancement layer streams. In practice, it is not necessary to

store the error signal for the 640x480 resolution. since fairly good quality video can be provided by bilinear interpolation of the 320x240 resolution images.

The base layer data is stored as a separate stream from the enhancement layer data on disk subsystem 90. This allows the system to admit more users when fewer users choose to receive the enhancement layer data. As will now be described, the base layer data is stored hierarchically, data for each frame being stored together. Each frame has a set of index planes corresponding to different number of bits used for the lookup.

The compressed stream comprises look-up indices with different number of bits depending on the bandwidth and quality requirement. The look-up indices for each frame are stored as groups of index planes pre-formatted with application level headers for network transmission. Preferably the four most significant bits of the lookup indices are stored together as the first section of the frame block. Then four additional 1-bit planes of look-up are stored in sequence, as separate sections of the frame block to provide lookup indices with 4, 5, 6, 7, 8 bits, respectively. The different lookup indices provide data streams with different bandwidth requirements.

With reference to Figure 1, server 20 fetches the base signal frame block from the disk 90, transmits the selected sections on the network 30, 30'. The re-packing of the bit planes into look-up indices is left to the receiving application at the client-end of the system.

The error data is placed similarly as another data stream. The look-up indices preferably are stored as the most significant two bits of the look-up indices in the first section for each frame block in the bit stream. Then follow the second two bits of the look-up indices as the second section, followed in turn by four additional 1-bit sections of lookup indices that are stored to provide look-up indices with 2, 4, 5, 6, 7, 8 bits, respectively. Other encoding bit patterns might instead be used, however.

Preferably the video server uses RAID-like techniques to stripe each (data stream) across several drives. RAID design techniques are known in the art; e.g., see F. Tobagi, et al., "Streaming RAID - A disk array management system for video files," Proc. ACM Multimedia 1993. A RAID design allows for recovery from failure of any single disk without diminishing the capacity of the server. A RAID design removes any restriction on the number of active users of a given video title, as long as the multiple users can be accommodated within the server total bandwidth. That is, the usage can range from all active users receiving the same title at different offsets to all receiving different streams.

The streams of base and enhancement layer data preferably are striped in fixed size units across the set of drives in the RAID group, with parity placed on an additional drive. The selection of the parity drive is fixed since data updates are quite rare compared to the number of times the streams are read. The preferred striping policy keeps all of the look-up indices for an individual frame together on one disk. This allows for ease of positioning when a user single steps or fast-forwards the user's display, although there is a penalty in some loss of storage capacity due to fragmentation. Use of parity on the stripe level allows for quick recovery after a drive failure at the cost of using substantially more buffer space to hold the full exclusive-OR recovery data set.

In this example, the video server utilizes the planar bit stream format directly as the basis for the packet stream in the network layer. The embedded stream bits plus the application packet header are read from disk 90 and are transmitted on the network in exactly the same format. For example, in the preferred embodiment the base video layer has the four most significant bits of the look-up indices stored together. Thus. those bits are transmitted as one 2440 byte packet, and each additional index bit plane of the less significant bits is transmitted as a separate 640 byte packet.

The header preferably contains a frame sequence number, nominal frame rate, size, a virtual time stamp, and a bit plane type specifier sufficient to make each packet an identifiable stand-alone unit. The server uses the self identifying header to extract each bit plane group packet from the striped frame data retrieved from the disk subsystem.

The server also uses the header sequence and rate information as a means to pace the network transmission and disk read requests. The server uses a feedback loop to measure the processing and delivery time costs of the disk reads and queueing the network packets for transmission. The server then uses these measures to schedule the next disk read and packet transmission activities to match the video stream frame rate (i.e., at X milliseconds in the future start transmitting the next frame of video). The server can moderate the transmission rate based on slow down/speed-up feedback from the decoder.

The receiving end of the embodiment will now be described with reference to Figure 1. At the receiving end, decoder(s) 40 include a central processing unit ("CPU") 140 that includes a CPU per se and associated memory including cache memory. Decoder(s) 40 further includes a mechanism 145 for synchronizing audio and video information from the incoming embedded stream, as well as audio and video decoders 150, 160. The output from these decoders is coupled to sound generator, e.g., a speaker, and to video displays 180.

Preferably the decoders are software-based, with the decoding process algorithm preferably are stored in memory, for example memory 140, for execution by the associated CPU. Alternatively, in applications where full CPU operations are not required, for example simple display applications, decoders according to the present invention may be implemented in hardware, e.g., in a simply CPU' and read-only memory ("ROM") unit 155. Within unit 155 is a relatively simple central processor unit CPU' that, collectively with the associated ROM, represents a hardware unit that may be produced for a few dollars.

Target decoder system 40 should be able to define at least 160x120. 320x240. 640x480 pixel spatial resolutions, and at least 1 to 30 frames per second temporal resolution. Decoder system 40 must also accommodate bandwidth scalability with a dynamic range of video data from 10 kbps to 10 kbps to 10 Mbps. In this arrangement, video encoder 60 provides a single embedded stream from which different streams at different spatial and temporal resolutions and different data rates can be extracted by decoders 40, depending on decoder capabilities and requirements. However, as noted, encoder embedding is independent of the characteristics of the decoder(s) that will receive the single embedded information stream.

For example, decoder 40 can include search engines that permit a user to browse material for relevant segments, perhaps news, that the user may then select for full review. Within server 20, video storage 90 migrates the full resolution, full frame rate news stories based on their age and access history from disk to CD ROM to tape, leaving lower resolution versions behind to support the browsing operation. If a news segment becomes more popular or important, the higher resolution can then be retrieved and stored at a more accessible portion of the storage hierarchy 90.

The decoder(s) may be software-based and merely use the indices from the embedded bit-stream to look-up from a codebook that is designed to make efficient use of the cache memory associated with the CPU unit 140. In this arrangement, video stream decoding is straightforward, and consists of loading the codebooks into the CPU cache memory, and performing look-ups from the stored codebook tables. In practice, the codebook may be stored in less than about 12 Kb of cache memory.

Video decoder 160 may be software-based and uses a Laplacian pyramid decoding algorithm, and preferably can support up to three spatial resolutions, i.e., 160x120 pixels, 320x240 pixels, and 640x480 pixels. Further, decoder 160 can support any frame rate, as the frames are coded independently by encoder 60.

The decoding methodology is shown in Figure 3. To decode a 160x120 pixel image, decoder 160 at method step 410 need only decompress the base layer 160x120 pixel image 260. The resultant image 430 is copied to video monitor (or other device) 180. APPENDIX 1, attached hereto, is a sample of decompression as used with the present invention.

To obtain a 320x240 pixel image, decoder 160 first decompresses (step 410) the base layer 260, and then at step 440 up-samples to yield an image 450 having the correct spatial resolution, e.g., 320x240 pixels. Next, at step 460, the error data in the first enhancement layer 340 is decompressed. The decompressed image 470 is then added at step 480 to up-sampled base image 450. The resultant 320x240 pixel image 490 is coupled by decoder 160 to a suitable display mechanism 180.

To obtain a 640x480 pixel image, the up-sampled 320x240 pixel image 450 is up-sampled at step 500 to yield an image 510 having the correct spatial resolution, e.g., 640x480 pixels. Next, at step 520, the error data in the second enhancement layer 400 is decompressed. The decompressed image 530 is added at step 540 to the up-sampled base image 510. The resultant 640x480 pixel image 550 is coupled by decoder 160 to a suitable display mechanism 180.

As seen from Figure 3 and the above-description, it will be appreciated that obtaining the base layer from the embedded bit stream requires only look-ups, whereas obtaining the enhancement layers involves performing look-ups of the base and error images, followed by an addition process. Preferably, the decoder is software-based and operates rapidly in that all decoder operations are actually performed beforehand, i.e., by preprocessing. The TSVQ decoder codebook contains the inverse DCT performed on the codewords of the encoder codebook. As noted, in applications such as video displays where a complex CPU 140 would not necessarily be present, the video decoder may be implemented in hardware, e.g., by storing the functions needed for decoding in ROM 155, or the equivalent. In practice, ROM 155 may be as small as about 12 Kb.

Thus, at the decoder there is no need for performing inverse block transforms. Color conversion, i.e., YUV to RGB, is also performed as a pre-processing step by storing the corresponding color converted codebook. To display video on a limited color palette display, the resulting codewords of the decoder codebook are quantized using a color quantization algorithm. One such algorithm has been proposed by applicant Chaddha et al., "Fast Vector Quantization Algorithms for Color Palette Design Based on Human Vision Perception," accepted for publication IEEE Transactions on Image Processing.

In this arrangement, color conversion involves forming a RGB or YUV color vector from the codebook codewords, which are then color quantizing to the required alphabet size. Thus, the same embedded index stream can be used for displaying images on different alphabet decoders that have the appropriate codebooks with the correct alphabet size, e.g., 1-bit to 24-bit color.

On the receiving end, the video decoder 40, 40' is responsible for reassembly of the lookup indices from the packets received from the network. If one of the less significant index bit plane packets is somehow lost, the decoder uses the more significant bits to construct a shorter look-up table index. This yields a lower quality but still recognizable image.

The use of separately identified packets containing index bit planes makes it possible for networks to easily scale the video as a side effect of dropping less important packets. In networks providing QOS qualifiers such as ATM, multiple circuits can be used to indicate the order in which packets should be dropped (i.e., the least significant bit plane packets first). In an IP router environment, packet filters can be constructed to appropriately discard less important packets. For prioritized networks, the base layer will be sent on the high priority channel while the enhancement layer

will be sent on the low priority channel. To provide error resiliency, using a fixed-rate coding scheme with some added redundancy allows robustness in the event of packet loss.

It will be appreciated that a server according to the present invention can support two usage scenarios: point-to-point demand (e.g., networks 30', decoders 40' in Figure 1), or multicast (e.g., network cloud 50, networks 30, decoders 40 in Figure 1).

In a point-to-point demand environment, each destination system decoder presents its specific requirements to the server. The server then sends the selected elements of the embedded stream across the network to the destination. A separate network stream per destination allows the user to have VCR style functionality such as play/stop/rewind fast forward/fast reverse. If congestion occurs on the network, the routers and switches can intelligently drop packets from the embedded stream to give a lesser number of lookup bits.

In a multicast environment, the server, which has no information about the destination decoders, outputs the entire embedded stream for the different resolutions and rates onto the network as a set of trees. In the preferred embodiment, there may be one to eleven trees, depending on the granularity of traffic control desired. The primary traffic management is performed during the construction of the unicast trees, by not adding branches of the trees carrying the less important bit streams to the lower bandwidth networks. The network in this case takes care of bandwidth mismatches by not forwarding packets to the networks which are not subscribed to a particular tree. Switches and routers can still react to temporary congestion by intelligently dropping packets from the embedded stream to deliver fewer bits of look-up.

The delivery system treats the audio track as a separate stream that is stored on disk 100 and transmitted across the network as a separate entity. The audio format supports multiple data formats from 8 KHz telephony quality (8 bit mu-law) to 48 KHz stereo quality audio (2 channel, 16 bit linear samples). In practice, many video clips may have 8 KHz telephone audio, to permit material distribution over medium-to-low bandwidth networks. The server can store separate high and low quality audio tracks, and transmit the audio track selected by the user. As the audio transits the network on a separate circuit, the audio can easily be given a higher QOS than the video streams. Rather than further load the networks with duplicate audio packets, as is known in the prior art, in the present invention the audio is ramped down to silence when packets are overly delayed or lost.

As the audio and video are delivered via independent mechanisms to the decoding system, the two streams must be synchronized by mechanism 145 for final presentation to the user. At the decoder, the receiving threads communicate through the use of a shared memory region, into which the sequence information of the current audio and video display units are written.

The human perceptual system is more sensitive to audio dropouts than to video drops, and audio is more difficult than video to temporarily reprocess. Thus, the decoder preferably uses the audio coder as the master clock for synchronization purposes. As the streams progress, the decoder threads post the current data items' sequence information onto a "blackboard" or scratchpad portion of memory associated with CPU unit 140. The slave threads (such as the video decoder) use the posted sequence information of the audio stream to determine when their data element should be displayed. The slave threads then delay until the appropriate time if the slave is early (e.g., more than 80 ms ahead of the audio). If the slave data is too late (e.g., more than 20 ms behind the audio), then it is discarded on the assumption that continuing to process late data will delay more timely data.

The video decoder can optionally measure the deviation from the desired data delay rate and send speed-up and slow-down indications back to the video server. This process synchronizes streams whose elements arrive in a timely fashion and does not allow a slow stream to impede the progress of the other streams.

In the even of scarcity of resources, some global prioritization of user requests must take place to guard against overload collapse. In a practical system, payment for services and resources may be used to define the overall value of each resource allocation decision. Given these values, a total ordering of the user requests can be made, e.g., by admission control 110, and the less important requests can be dropped. The user specifies what he or she is willing to pay for a given service. This proposed amount and the required associated resources (network and disk bandwidth) are submitted to an electronic market, e.g., admission control 110, which uses micro-economic models to decide what amount of bandwidth resource is available to the user. Such techniques are known in the art, e.g., M. Miller, "Extending markets inward," Bionomics Conference, San Francisco, California (Oct. 1994).

For the particular bandwidth required, a table is indexed to find the best possible combination of spatial resolution, frame rate and data rate (number of bits of look-up to be used) to give the best quality of decompressed video. Preferably such table is built using a subjective distortion measure, such as described by N. Chaddha and T. H. Y. Meng, "Psycho-visual based distortion measures for image and video compression", Proc. of Asilomar Conference on Signals, Systems and Computers, Nov. 1993. Preferably, the user also has the option of specifying the spatial resolution, frame rate and bandwidth directly.

It will be appreciated that the described overall system combines a software-based encoder with an encoding compression algorithm, disk management, network transport, software-based decoder, and synchronization mechanism to provide an end-to-end scalable video delivery service. The service may be divided into three groups of components, comprising preprocessing, media server, and media player.

The processing components include audio capture, video capture, video compression, and a data stripping tool.

The video is captured and digitized using single step VCR devices. Each frame is then compressed off-line (non-real time) using the encoding algorithm. At present, it takes about one second on a SparcStation 20 Workstation to compress a frame of video data, and single step VCR devices can step at a one frame per second rate permitting overlap of capture and compression.

The audio data preferably is captured as a single pass over the tape. The audio and video time stamps and sequence numbers are aligned by the data striping tool as the video is stored to facilitate later media synchronization. The audio and video data preferably are striped onto the disks with a user-selected stripe size. In a preferred embodiment, all of the video data on the server uses a 48 kilobyte stripe size, as 48 kilobytes per disk transfer provides good utilization at peak load with approximately 50% of the disk bandwidth delivering data to the media server components.

The media server components include a session control agent, the audio transmission agent, and the video transmission agent. The user connects to the session control agent on the server system and arranges to pay for the video service and network bandwidth. The user can specify the cost he/she is willing to pay and an appropriately scaled stream will be provided by the server. The session control agent (e.g., admission control mechanism 110) then sets up the network delivery connections and starts the video and audio transmission agents. The session control agent 110 is the single point of entry for control operations from the consumers remote control, the network management system, and the electronic market.

The audio and video transmission agents read the media data from the striped disks and pace the transmission of the data onto the network. The video transmission agent scales the embedded bit-stream in real-time by transmitting only the bit planes needed to reconstruct the selected resolution at the decoder. For example, a 320x240 stream with 8 bits of base, 4 bits of enhancement signal at 15 frames per second will transmit every other frame of video data with all 5 packets for each frame of the base and only two packets containing the four most significant bits of the enhancement layer resulting in 864 Kb of network utilization. The server sends the video and audio either for a point-to-point situation or a multicast situation.

The media player components are the software based video decoder 40, 40', the audio receiver, and a user interface agent. The decoder receives the data from the network and decodes it using look-up tables and places the results onto the frame buffer. The decoder can run on any modern microprocessor unit without the CPU loading significantly. The audio receiver loops reading data from the network and queuing up for the data for output to the speaker. In the event of audio packet loss, the audio receiver will ramp the audio level down to silence level and then back up to the nominal audio level of the next successfully received audio packet. The system performs media synchronization to align the audio and video streams at the destination, using techniques such as described by J. D. Northcutt and E. M. Kuerner, "System Support for Time-Critical applications," Proc. NOSSDAV 91, Germany, pp 242-254.

End-to-end feedback is used in the on demand case to control the flow. In the multicast case, the destinations are slaved to the flow from the server with no feedback. The user interface agent serves as the control connection to the session agent on the media server passing flow control feedback as well as the user's start/stop controls. The user can specify the cost he or she is willing to pay and an appropriate stream will be provided by the system.

A prototype system embodying the present invention uses a video data rate that varies from 19.2 kbps to 2 Mbps depending on the spatial and temporal requirement of the decoder and the network bandwidth available. The PSNR varies between 31.63 dB to 37.5 dB. Table 1 gives the results for the decoding of a 160x120 resolution video on a SparcStation 20. It can be seen from Table 1 that the time required to get the highest quality stream (8-bit index) at 160x120 resolution is 2.45 ms per frame (sum of lookup and packing time). This corresponds to a potential frame rate of 400 frames/sec.

TABLE 1.

| RESULTS FOR 160x120 RESOLUTION (DECODER) | | | | |
|---|---|---|---|---|
| No. of Bits of Lookup | PSNR (dB) | Bandwidth as a function of frame rate (N) Kbps | CPU time per frame (ms) | Packing time per frame (ms) |
| 4 | 31.63 dB | 19.2N | 1.24 ms | 0 ms |
| 5 | 32.50 dB | 24N | 1.32 ms | 0.52 ms |
| 6 | 34 dB | 28.8N | 1.26 ms | 0.80 ms |
| 7 | 35.8 dB | 33.6N | 1.10 ms | 1.09 ms |
| 8 | 37.2 dB | 38.4N | 1.18 ms | 1.27 ms |

Similarly, Table 2 gives the results for the decoding of a 320x240 resolution video on a SparcStation 20. It can be seen from Table 2 that the time required to get the highest quality stream (8-bit base index and 8-bit first enhancement layer index) at 320x240 resolution is 7.76 ms per frame (sum of look-up and packing time). This corresponds to a potential frame rate of 130 frames/sec.

### TABLE 2. RESULTS FOR 320x240 RESOLUTION (8 BIT-LOOKUP BASE)

| No. of Bits of Lookup | PSNR (dB) | Bandwidth as a function of frame rate (N) Kbps | CPU time per frame (ms) | Packing time per frame (ms) |
|---|---|---|---|---|
| 2 | 33.72 dB | 48N | 6.01 ms | 0.385 ms |
| 4 | 35.0 dB | 52.8N | 6.04 ms | 0.645 ms |
| 5 | 35.65 dB | 62.4N | 6.05 ms | 0.92 ms |

| | | | | |
|---|---|---|---|---|
| 6 | 36.26 dB | 67.2N | 6.08 ms | 1.20 ms |
| 7 | 36.9 dB | 72N | 6.04 ms | 1.48 ms |
| 8 | 37.5 dB | 76.8N | 6.09 ms | 1.67 ms |

Table 3 gives the results for the decoding of a 640x480 resolution video again on a SparcStation 20. It can be seen from Table 3 that the time required to get the highest quality stream (8-bit base and 8-bit enhancement layer) at 640x480 resolution is 24.62 ms per frame (sum of lookup and packing time). This corresponds to a potential frame rate of 40 frames/sec.

### TABLE 3.

| RESULTS FOR 640x480 WITH 320x240 INTERPOLATED | | | | |
|---|---|---|---|---|
| No. of Bits of Lookup | PSNR (dB) | Bandwidth as a function of frame rate (N) Kbps | CPU time per frame (ms) | Packing time per frame (ms) |
| 2 | 33.2 dB | 48N | 22.8 ms | 0.385 ms |
| 4 | 34 dB | 52.8N | 22.87 ms | 0.645 ms |
| 5 | 34.34 dB | 62.4N | 23.14 ms | 0.92 ms |
| 6 | 34.71 dB | 67.2N | 22.93 ms | 1.20 ms |
| 7 | 35.07 dB | 72N | 22.90 ms | 1.48 ms |
| 8 | 35.34 dB | 76.8N | 22.95 ms | 1.67 ms |

Table 4 shows the results for each individual disk for 160x120 resolution video. It can be seen that to get the highest quality stream (8-bit base) at 160x120 requires 5.60 ms of CPU time and an average CPU load of 2% on a SparcStation 20 workstation. The average disk access time per frame is 16 ms.

## TABLE 4. RESULTS FOR 160x120 AT THE DISK SERVER

| No. of Bits of Lookup | Bandwidth as a function of frame rate (N) Kbps | CPU time per frame (ms) | Seek-time (ms) | Avg. CPU Load |
|---|---|---|---|---|
| 4 | 19.2N | 2.84 ms | 16 ms | 1 % |
| 5 | 24N | 3.67 ms | 16 ms | 1 % |
| 6 | 28.8N | 4.48 ms | 14 ms | 2 % |
| 7 | 33.6N | 4.92 ms | 14 ms | 2 % |
| 8 | 38.4N | 5.60 ms | 16 ms | 2 % |

Similarly, Table 5 shows the results for each individual disk for 320x240 resolution video. It can be seen that obtaining the highest quality stream (8-bit base and 8-bit enhancement layer) at 320x240 requires 12.73 ms of CPU time and an average CPU load of 7% on a SparcStation 20 workstation. The average disk access time per frame is 18 ms.

## TABLE 5.

| RESULTS FOR 320x240 AT THE DISK SERVER | | | | |
|---|---|---|---|---|
| No. of Bits of Lookup | Bandwidth as a function of frame rate (N) Kbps | CPU time per frame (ms) | Seek-time (ms) | Avg. CPU Load |
| 2 | 48N | 10.47 ms | 18 ms | 6 % |
| 4 | 52.8N | 11.02 ms | 16 ms | 6 % |
| 5 | 62.4N | 11.55 ms | 18 ms | 6 % |
| 6 | 67.2N | 12.29 ms | 20 ms | 7 % |
| 7 | 72N | 12.55 ms | 20 ms | 7 % |
| 8 | 76.8N | 12.73 ms | 18 ms | 7 % |

```
/*
 * Scalable Video Displayer — Main Program
 * Copyright 1995 Sun Microsystems, Inc.
 */


typedef signed int errorcode;

pixel *baselookupcodebook;
errorcode *errorcodebook;

errorcode *largecodebook;

/*
 * invert the single eight bit base index into the base 2x2 codeboo.
 * and store into the destination image
 */

static void
BasePixelInvert(pixel *dest, int stride, int index)
{
        index <<= 2; /* index into the 4 pixel chunk */

        *dest = baselookupcodebook[index];          /* the first pixel on line 0 */
        *(dest+1) = baselookupcodebook[index+1]; /* the second pixe. on line 0 */

        *(dest+stride) = baselookupcodebook[index+2]; /* the first }ixel on line 1 */
        *(dest+stride+1) = baselookupcodebook[index+3]; /* the secoid pixels on line 1 */
}


/*
 * step through the input indicies and
 * invert each eight bit index into the base 2x2 codebook
 * and store into the destination image at the
 * correct location
 * each input index gives 4 destination pixels
 */


void
BaseInvert(int nv, int nh, pixel *image, int stride, unsigned char *source)
{
        int x, y;
        int index;
        unsigned char *dest;



        for (y = nv; y > 0; —y) {
                dest = image;
                for (x = nh; x > 0; —x) {
                        index = *source++;
                        BasePixelInvert(dest, stride, index);
                        dest += 2; /* step to the next destination rixel */
                }
                image += 2 * stride;     /* step to the start of the next two lines */
        }
}

/*
 * invert the eight bit base index values into the 2x2 base value
 * and combine with the results of inverting the eight bit error
 * index values 4x4 error difference.
 */
```

```c
static void
ErrorPixelInvert(pixel *dest, int stride, pixel base_pixel, int error_index)
{
        /* each destination base pixel gets combined with four error values */

        *(dest) = base_pixel +  errorlookupcodebook[error_index];
        *(dest+1) = base_pixel +  errorlookupcodebook[error_index+1];

        *(dest+stride) = base_pixel +  errorlookupcodebook[error_index+4];
        *(dest+stride+1) = base_pixel +  errorlookupcodebook[error_index+5];
}

/*
 * step through the input indicies and
 * invert each eight bit base index into the base 2x2 codebook
 * and invert each eight bit error index into the error 4x4 codebook
 * combine the results and then store into the destination image
 * at the correct location
 *
 * each pair of input indicies gives 16 destination pixels
 */


ErrorInvert(nv, nh, image, stride, source, errorband)
int nv;
int nh;
pixel *image;
int stride;
unsigned char *source;
unsigned char *errorband;
{
        unsigned char *line1, *line3;
        int base_index, error_index;
        pixel base_pixel;


        while (nv > 0) {
                --nv;
                line1 = image;
                line3 = line1 + (2 * stride);

                for (x = nh; x > 0; --x) {

                        /* index into 4 pixel chunk for base 2x2*/
                        base_index = *source++;
                        base_index <<= 2;

                        /* index into the 16 pixel chunk error 4x4*/
                        error_index = *errorband++;
                        error_index <<= 4;

                        base_pixel = baselookupcodebook[base_index+0];
                        ErrorPixelInvert(line1, stride, base_pixel, error_index);
                        line1 += 2;
                        error_index += 2;

                        base_pixel = baselookupcodebook[base_index+1];
                        ErrorPixelInvert(line1, stride, base_pixel, error_index);
                        line1 += 2;
                        error_index += 6;

                        base_pixel = baselookupcodebook[base_index+2];
                        ErrorPixelInvert(line3, stride, base_pixel, error_index);
                        line3 += 2;
```

```
                        error_index += 2;

                        base_pixel = baselookupcodebook[base_index+:];
                        ErrorPixelInvert(line3, stride, base_pixel, error_index);
                        line3 += 2;
                }
                image += 4 * stride; /* step to next set of four lines */
        }
}

display_agent()
{
        int nbase;
        int nerror;
        unsigned char *basedata;
        unsigned char *errordata;

        while (global_status != EXIT) {
                nbase = AssembleBasePacketsFromNetwork(&basedata);

                SetBaseCodeBook(nbase);

                nerror = AssembleErrorPacketsFromNetwork(&errordata);

                SetErrorCodeBook(nbase);

                if (large_display)
                        ErrorInvert(120, 160, Ximage->data, ximage_width, basedata, errordat
                else
                        BaseInvert(120, 160, Ximage->data, ximage_width, basedata);

                /* use standard X11 put image to display the result */

                XPutImage(display, xid, gc, Ximage, 0, 0, 0, 0, ximage_width , ximage_heigh
        }
}
```

Claims

1. For use in a video delivery system server having a source of video images, an encoder providing an embedded bit stream containing information including image data at at least two spatial resolutions, said embedded bit stream being sent over at least one network to at least one decoder, the encoder including:

   a central processor unit coupled to a memory unit;
   encoding means, digitally stored in said memory unit and coupled to said source of video images and receiving a first image at first spatial resolution, for decimating said first image to form an first intermediate image at half said highest resolution, for decimating said first intermediate image to form a second intermediate image, for compressing said second intermediate image to form a base layer image whose resolution is less than said first image;
   said encoding means further decompressing said base layer image to form a third intermediate image, and interpolating said third intermediate image to form a fourth intermediate image, and for subtracting said fourth intermediate image from said first intermediate image to form a fifth intermediate image, and for compressing said fifth intermediate image to form a first enhancement layer image whose resolution is less than said first image but greater than said base layer image;
   said embedded bit stream containing at least said base layer image and said first enhancement layer image.

2. The encoder of claim 1, wherein said embedded bit stream contains at least three spatial resolutions including an additional image, whose resolution equals that of said first image, and said base layer image, and wherein said encoding means further interpolates said fourth intermediate image to form a sixth intermediate image that is subtracted from said first image to form a seventh intermediate image that is compressed to form a second enhancement layer image whose resolution equals that of said first image;
   said embedded bit stream further containing said second enhancement layer image.

3. The encoder of claim 1, wherein said embedded bit stream includes spatial resolution data encoded in pixel blocks, and wherein said encoding means encodes said spatial resolution data using a discrete cosine transformation followed by a tree-structured vector quantization upon results of said transformation.

4. The encoder of claim 2, wherein said first image has a resolution of 640x480 pixels, said additional image has a resolution of 320x240 pixels, and said base layer image has a resolution of 160x120 pixels.

5. The encoder of claim 4, wherein said embedded bit stream includes spatial resolution data encoded in pixel blocks of size 2x2 bits for said base layer image, of size 4x4 bits for said additional image, and of size 8x8 bits for said first image, and wherein said encoding means encodes said spatial resolution data using a discrete cosine transformation followed by a tree-structured vector quantization upon results of said transformation.

6. The encoder of claim 3, wherein transform coefficients include input-weighted squared error defined as follows:

$$d_I(y,\hat{y}) \;=\; \sum_{j=1}^{K} w_j (y_j - \hat{y}_j)^2$$

where $y_j$ and $\hat{y}_j$ are components of a transformed vector $y$ and of a corresponding reproduction vector $\hat{y}$, and where $w_j$ is a component of a weight vector generally dependent only upon $y$.

7. The encoder of claim 6, wherein said tree-structured vector quantization includes a perception model.

8. The encoder of claim 7, wherein said weight vector components reflect human visual sensitivity to quantization errors in different transform coefficients.

9. The encoder of claim 7, wherein said tree-structured vector quantization has a tree depth $R$ and has a vector dimension is $k$, and wherein bitstream bit rates $O/k, \ldots\ldots, R/k$ are provided.

10. The encoder of claim 7, wherein indices are transmitted in said embedded stream with different index planes;

a first index plane containing a first index for a rate 1/*k* tree-structured vector quantization reference, and
a second index plane containing a second index for a rate 2/*k* tree-structured vector quantization reference.

11. The encoder of claim 10, wherein said embedded bit stream includes data packets, and wherein said indices are associated with a relative priority of importance of at least some of said data packets.

12. The encoder of claim 11, wherein said embedded bit stream is sent over a network having throughput insufficient for decoder receipt of a highest spatial image,
    wherein said relative priority associated with said data packets permits selective non-transmission of relatively unimportant packets.

13. The encoder of claim 11, wherein data for each video frame is stored together, and wherein each frame has an associated set of index planes with associated packet headers.

14. The encoder of claim 1, wherein said encoding means further includes at least one option selected from the group consisting of (a) estimation of motion associated with a transmitted image, and (b) conditional replenishment of a said image.

15. A method, for use in a video delivery system server having a source of video images, of encoding an embedded bit stream containing information including image data at at least two spatial resolutions, said embedded bit stream being sent over at least one network to at least one decoder, the method including the following steps:

    (a) providing a central processor unit coupled to a memory unit; and
    (b) providing encoding means, digitally stored in said memory unit and coupled to said source of video images and receiving a first image at first spatial resolution, for decimating said first image to form an first intermediate image at half said highest resolution, for decimating said first intermediate image to form a second intermediate image, for compressing said second intermediate image to form a base layer image whose resolution is less than said first image;

    said encoding means further decompressing said base layer image to form a third intermediate image, and interpolating said third intermediate image to form a fourth intermediate image, and for subtracting said fourth intermediate image from said first intermediate image to form a fifth intermediate image, and for compressing said fifth intermediate image to form a first enhancement layer image whose resolution is less than said first image but greater than said base layer image;
    said embedded bit stream containing at least said base layer image and said first enhancement layer image.

16. The method of claim 15, wherein said embedded bit stream contains at least three spatial resolutions including an additional image, whose resolution equals that of said first image, and said base layer image, and wherein at step (b), said encoding means further interpolates said fourth intermediate image to form a sixth intermediate image that is subtracted from said first image to form a seventh intermediate image that is compressed to form a second enhancement layer image whose resolution equals that of said first image;
    said embedded bit stream further containing said second enhancement layer image.

17. The method of claim 15, wherein said embedded bit stream includes spatial resolution data encoded in pixel blocks, and wherein at step (b), said encoding means encodes said spatial resolution data using a discrete cosine transformation followed by a tree-structured vector quantization upon results of said transformation.

18. The method of claim 17, wherein at step (b), transform coefficients include input-weighted squared error defined as follows:

$$d_{\underline{I}}(y, \hat{y}) = \sum_{j=1}^{K} w_j (y_j - \hat{y}_j)^2$$

where $y_j$ and $\hat{y}_j$ are components of a transformed vector $y$ and of a corresponding reproduction vector $\hat{y}$, and where

$w_j$ is a component of a weight vector generally dependent only upon $y$.

19. The method of claim 18, wherein step (b) includes providing said weight vector components that reflect human visual sensitivity to quantization errors in different transform coefficients.

20. The method of claim 18, wherein at step (b), said tree-structured vector quantization has a tree depth $R$ and has a vector dimension is $k$, and wherein bitstream bit rates $O/k$,......., $R/k$ are provided.

21. The method of claim 18, wherein said embedded bit stream includes data packets, and wherein at step (b), indices are transmitted in said embedded stream with different index planes;

   a first index plane containing a first index for a rate $1/k$ tree-structured vector quantization reference, and
   a second index plane containing a second index for a rate $2/k$ tree-structured vector quantization reference;
   said indices being associated with a relative priority of importance of at least some of said data packets.
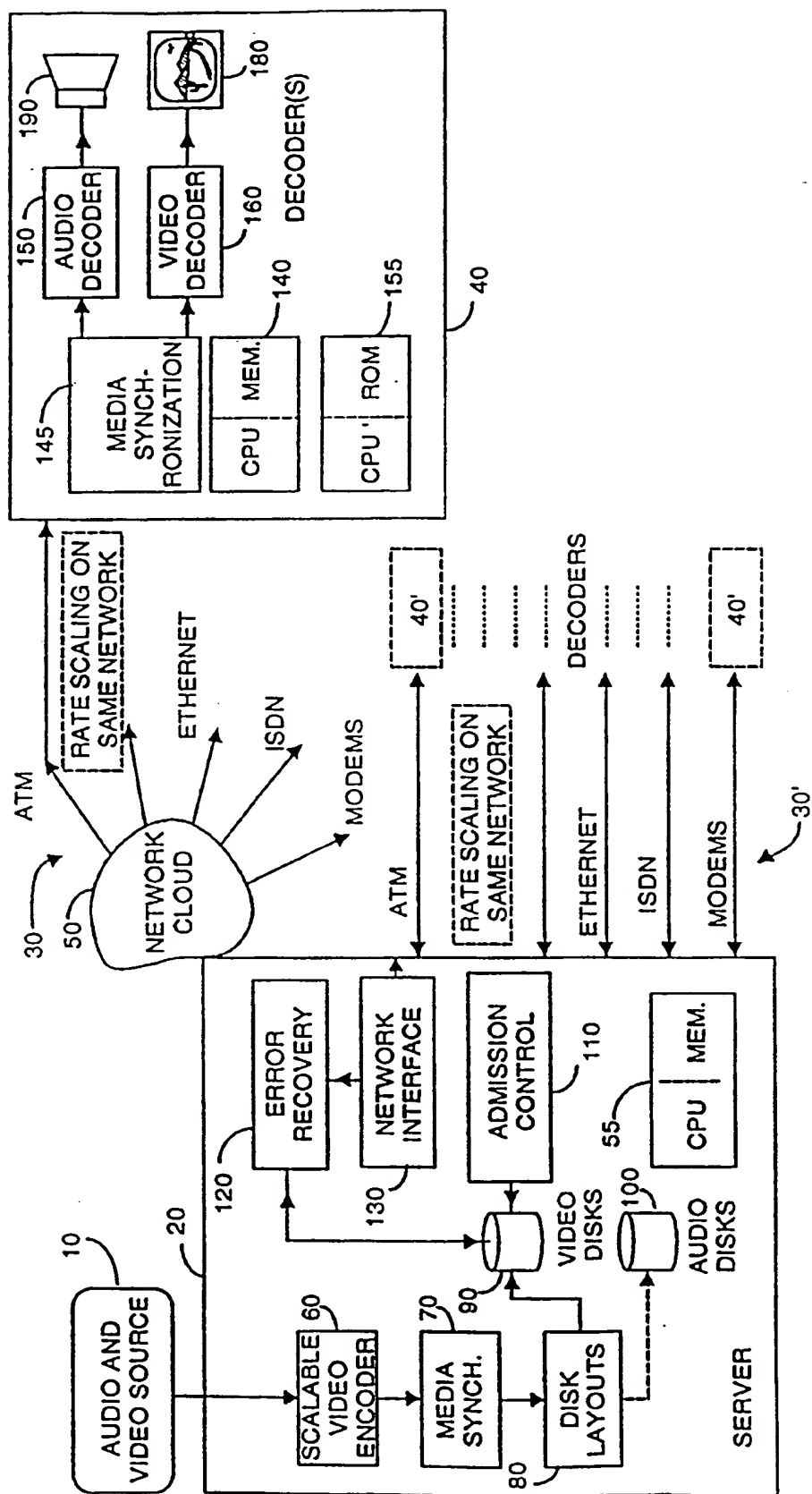
FIGURE 1
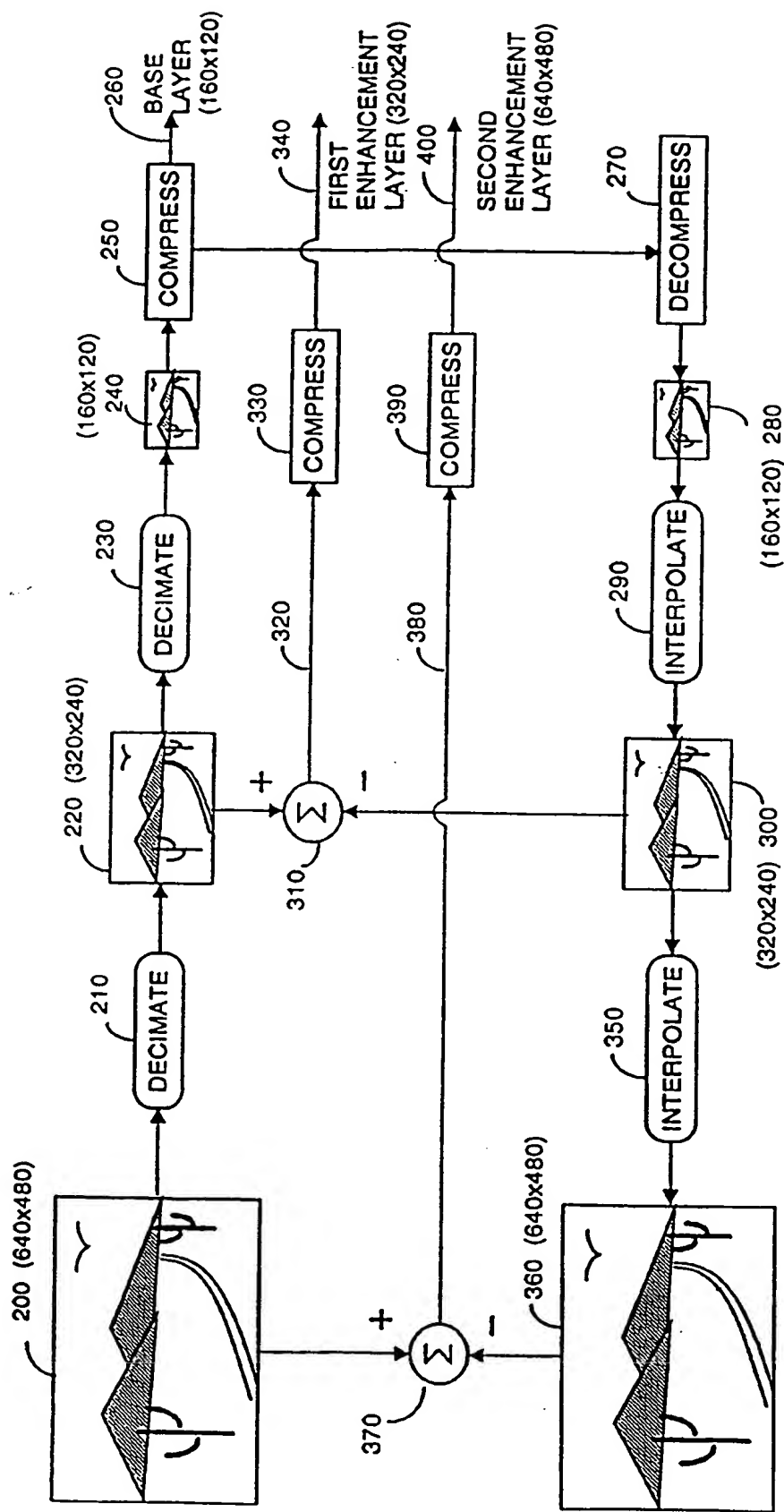
FIGURE 2
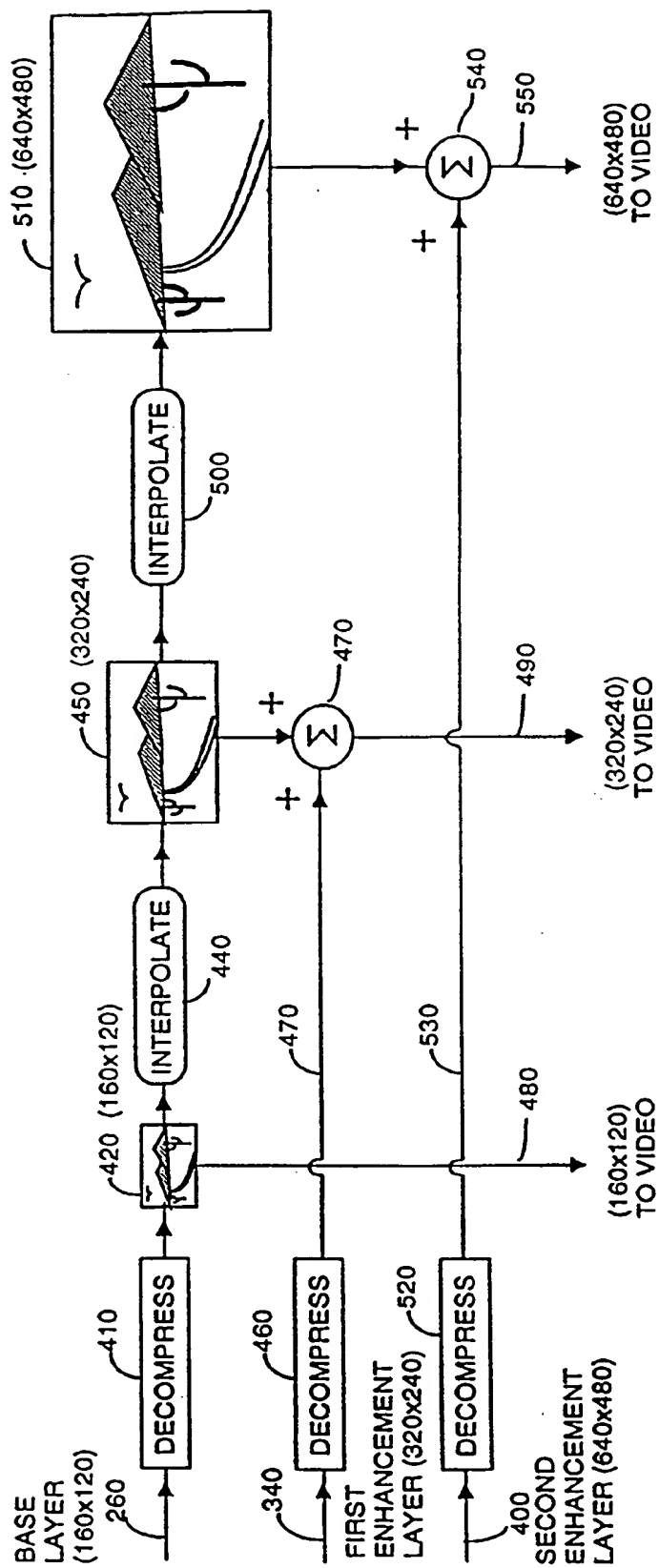
**FIGURE 3**

## (12) EUROPEAN PATENT APPLICATION

(72) Inventors:
• Chaddha, Navin
Stanford, California 94305 (US)

• Northcutt, J. Duane, c/o Sun Microsystems, Inc.
Mountain View, CA 94043-1109 (US)
• Wall, Gerald A.
San Jose, California 95136 (US)
• Hanko, James G.
Redwood City, California 94061 (US)

(74) Representative: Cross, Rupert Edward Blount
BOULT WADE TENNANT,
27 Furnival Street
London EC4A 1PQ (GB)

### (54) Encoder for an end-to-end scalable video delivery system

(57) A software-based encoder is provided for an end-to-end scalable video delivery system that operates over heterogeneous networks. The encoder utilizes a scalable video compression algorithm based on a Laplacian pyramid decomposition to generate an embedded information stream. The decoder decimates a highest resolution original image, e.g. 640x480 pixels, to produce an intermediate 320x240 pixel image that is decimated to produce an intermediate 160x120 pixel image that is compressed to form an encodable base layer 160x120 pixel image. This base layer image is decompressed to form an image that is up-sampled by interpolation to produce an up-sampled 320x240 pixel image. This up-sampled image is subtracted from the intermediate 320x240 pixel image to form an error image that is compressed and encoded as a first enhancement 640x480 pixel layer. The decompressed base layer image is also up-sampled at step to produce an up-sampled 640x480 pixel image that is subtracted from the original 640x480 pixel image 200 to yield an error image that is compressed to yield a second enhancement 320x240 pixel layer. Collectively, the base and enhancement layers comprise the transmitted embedded bit stream. At the receiving end, the decoder extracts from the embedded stream different streams at different spatial and temporal resolutions. Because decoding requires only additions and look-ups from a small stored table, decoding occurs in real-time.
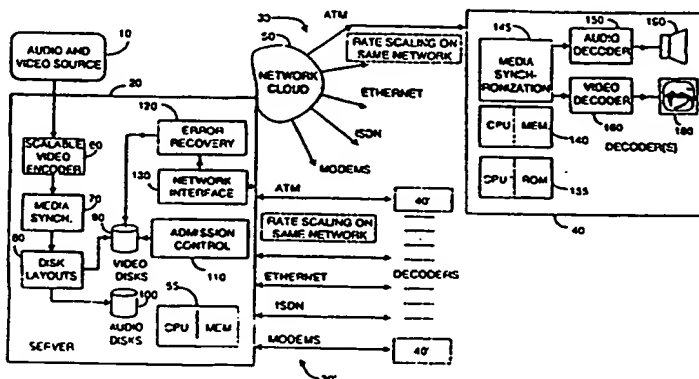
FIGURE 1

## DOCUMENTS CONSIDERED TO BE RELEVANT

| Category | Citation of document with indication, where appropriate, of relevant passages | Relevant to claim | CLASSIFICATION OF THE APPLICATION (Int.Cl.6) |
|---|---|---|---|
| X | SEOP HYEONG PARK ET AL: "A PYRAMID IMAGE CODER USING CLASSIFIED TRANSFORM VECTOR QUANTIZATION" SIGNAL PROCESSING EUROPEAN JOURNAL DEVOTED TO THE METHODS AND APPLICATIONS OF SIGNAL PROCESSING, vol. 22, no. 1, 1 January 1991, pages 25-42, XP000201121 | 1-4, 15-17 | H04N7/26 |
| Y | * abstract * | 6-8,18, 19 | |
| A | * section 3: Progressive CVQ for Laplacian planes in the DCT domain, pages 30-34 * <br><br> * figures 4,5 * | 5,9-14, 20,21 | |
| X | US 5 331 414 A (GOLIN STUART J) 19 July 1994 <br> * abstract * <br> * column 4, line 43 - column 4, line 60 * <br> * column 7, line 40 - column 12, line 32 * <br> * figures 2,5A,5B * | 1,2,4, 14-16 | |
| | | | TECHNICAL FIELDS SEARCHED (Int.Cl.6) |
| Y | US 5 282 031 A (KIM YONG-KYU) 25 January 1994 | 6-8,18, 19 | H04N |
| A | * column 2, line 16 - column 3, line 5 * <br><br> * column 4, line 1 - column 4, line 58 * <br> * figures 1,5 * | 1,3,15, 17 | |
| | -/-- | | |

The present search report has been drawn up for all claims

| Place of search | Date of completion of the search | Examiner |
|---|---|---|
| THE HAGUE | 30 June 1998 | Hampson, F |

## DOCUMENTS CONSIDERED TO BE RELEVANT

| Category | Citation of document with indication, where appropriate, of relevant passages | Relevant to claim | CLASSIFICATION OF THE APPLICATION (Int.CL6) |
|---|---|---|---|
| A | COSMAN P C ET AL: "TREE-STRUCTURED VECTOR QUANTIZATION WITH INPUT-WEIGHTED DISTORTION MEASURES" VISUAL COMMUNICATION AND IMAGE PROCESSING '91: VISUAL COMMUNICATION, BOSTON, NOV. 11 - 13, 1991, vol. PART 1, no. VOL. 1605, 11 November 1991, KOU-HU TZOU;TOSHIO KOGA, pages 162-171, XP000479227 * the whole document * | 1-8, 15-19 | |
| A | EP 0 367 264 A (SANYO ELECTRIC CO) 9 May 1990 * the whole document * | 1-21 | |

TECHNICAL FIELDS
SEARCHED    (Int.CL6)

The present search report has been drawn up for all claims

| Place of search | Date of completion of the search | Examiner |
|---|---|---|
| THE HAGUE | 30 June 1998 | Hampson, F |

This Page Blank (uspto)